Vers des interfaces graphiques flexibles de configuration

Simon Urli, Guillaume Perez, Heytem Zitoun, Mireille Blay-Fornarino, Philippe Collet, Philippe Renevier-Gonin

Vers des interfaces graphiques flexibles de configuration

Simon Urli, Guillaume Perez, Heytem Zitoun, Mireille Blay-Fornarino, Philippe Collet, Philippe Renevier-Gonin

Laboratoire I3S (UNS - CNRS)
Les Algorithmes - Bât Euclide B
2000 route des Lucioles – B.P. 121
F-06903 Sophia Antipolis Cedex

prenom.nom@unice.fr

RÉSUMÉ. *Par leur utilisation dans des domaines maintenant très divers, les lignes de produits logiciels sont de plus en plus confrontées à une forte évolution couplée à des besoins de configuration par des utilisateurs finaux. L'interface graphique de configuration doit alors être déduite, de manière générique, des modèles de variabilité de la ligne, mais les travaux relatifs aux IHM soulignent les faiblesses ergonomiques de ce type d'approche. Dans cet article, nous relatons une expérience de réalisation d'une interface de configuration dédiée à la création de systèmes de diffusion d'informations, basée sur une approche mixte mêlant généricité et adaptation de l'interface graphique par annotation des feature models décrivant la variabilité.*

ABSTRACT. *With their important usage in many domains, Software Product Lines (SPL) are more and more facing frequent evolution coupled with the need for configuration by end users. The configuration User Interface is expected to be deduced, in a generic way, from the variability models in the SPL. But previous work in HCI denote the ergonomic weakness of automatic-generated User Interface. In this paper, we report on the development of a graphical user interface for configuration of an information broadcasting systems SPL. Our method is a mix of genericity and adaptability of user interface by enhancing feature models with annotations.*

MOTS-CLÉS : *feature model, configuration, généricité, IHM*

KEYWORDS: *feature model, configuration, genericity, HCI*

1. Introduction

La force d'une ligne de produits est d'abstraire dans les termes du métier la production de produits en masquant à l'utilisateur final la mise en œuvre concrète de ses produits. L'usage d'une ligne de produits est d'autant plus pertinente que les produits potentiels présentent une grande variabilité et sont donc complexes à concevoir. L'interface de configuration/instanciation apparaît alors à l'homme du métier comme la part fondamentale de la ligne qui le conduira ou non à l'utiliser.

La mise en place des outils graphiques supportant la configuration/instanciation des produits pour une ligne donnée est donc essentielle à l'acceptation de la ligne par les utilisateurs finaux. Or, la construction des interfaces graphiques augmente de manière conséquente le coût de la ligne et exige des efforts importants d'appréhension des exigences ergonomiques liées au métier associé. De plus si la ligne est appelée à évoluer, il conviendra de prévoir l'évolution également des interfaces graphiques de configuration. La construction de la ligne dans un processus agile amplifie alors le problème puisque l'ajout de nouveaux features[1] dans la ligne peut impacter grandement les interfaces graphiques associées.

Bien que dans (Bosch *et al.,* 2001), la construction d'une interface dédiée est conseillée dans le contexte de lignes stables où peu de nouveaux features sont ajoutés, notre contexte fortement évolutif nous amène à considérer la génération automatique d'interface graphique de configuration.

Cependant, différents travaux dans le domaine des IHM dans les années 90 - début des années 2000 (Myers *et al.,* 2000) ont montré que la génération automatique d'interface graphique à partir de modèle(s) ne donnait pas de résultats aussi bons qu'un développement spécifique. Dans les démarches centrées utilisateurs (Coutaz, 1990), différents modèles sont pris en considération pour la réalisation d'une interface graphique : les concepts du domaine (ici les features models), le modèle utilisateur (qui va utiliser), le modèle de tâches (comment l'utilisateur va intervenir sur les concepts du domaine), le modèle d'interaction (comment représenter les concepts du domaine, comment les manipuler), etc. La difficulté principale pour la génération automatique de configurateur est la prise en compte de l'ensemble de ces modèles.

Dans (Boucher *et al.,* 2012), certains de ces modèles sont intégrés dans la génération de l'interface graphique de configuration, comme une représentation du modèle des tâches via le *Feature Configuration Workflows*. De manière complémentaire, nous proposons de prendre en compte l'utilisateur final et sa perception des produits à créer pour générer une interface graphique de configuration qui vérifie la propriété d'adaptabilité des IHM (Bastien *et al.,* 1993).

Cet article relate l'expérience menée dans le projet ANR Emergence YourCast, qui est présenté dans la partie 2. L'étude menée nous a permis d'identifier des élé-

1. Nous utiliserons dans l'ensemble du document les termes anglais *feature model* et *feature* afin de ne pas créer de contresens.

ments graphiques distinctifs entre les lignes utilisées (cf. partie 3). Sur la base de ces constatations nous proposons une approche, décrite dans la partie 4, qui repose sur la génération de l'interface de configuration à partir d'annotations sur les feature models, cette interface communiquant par la suite de manière synchrone avec un moteur de raisonnement sur les feature models. Nous discutons aussi les travaux connexes en partie 5, avant de conclure cet article.

2. Etude de cas : configuration de produits dédiés à la diffusion d'informations

2.1. *YourCast*

YourCast a pour but de faciliter la création de systèmes de diffusions d'informations. Il a pour vocation d'être utilisé par de nombreux acteurs non-informaticiens possédant une connaissance limitée du domaine des systèmes de diffusion.

Ce domaine regroupe des concepts aussi divers que :

– les *sources* d'informations : les services renvoyant les informations brutes souhaitées,

– les *renderers* : les opérateurs de traitement des informations pour leur donner une forme,

– les *layouts* : les structures d'écrans intégrant différentes zones,

– les *zones* : les emplacements dans lesquels les informations envoyées par les renderers seront affichées,

– les *behaviours* : le comportement qu'une zone doit adopter pour passer d'une information à une autre.

Tous ces concepts sont exprimés sous la forme de feature models indépendants, afin d'obtenir les différents produits voulus qu'il sera possible par la suite de combiner pour réaliser un système de diffusion fonctionnel. Ainsi, nous avons dès lors besoin d'une interface de configuration apte à prendre en compte ces différents feature models et à les rendre configurables par une personne non-experte.

La figure 1 présente une capture d'un système en cours de diffusion. On peut constater sur cet écran la présence d'un layout contenant deux zones, une centrale et une placée en bas de l'écran, diffusant deux sources différentes : la météo dans la zone centrale et des tweets dans la zone du bas. Les behaviours représentant la manière dont les informations changent ne sont évidemment pas visibles sur cette capture statique.

2.2. *Agilité du développement de la ligne*

La rapide évolution des besoins et des produits dans ces différents concepts force la prise en compte d'une grande évolutivité des caractéristiques. Nous visons par ailleurs

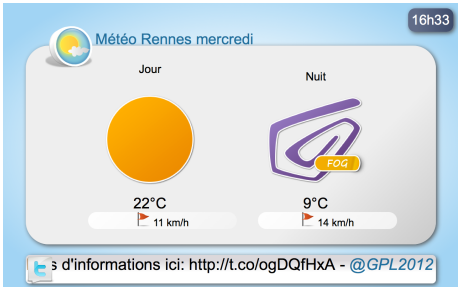

Figure 1. *Capture d'écran d'un système de diffusion déployé lors du GDR-GPL 2012 à Rennes*

la création d'une communauté de développeurs[2] et d'utilisateurs travaillant de concert, ce qui renforce le besoin d'une gestion incrémentale de l'ajout de nouveaux produits dans la ligne (Urli *et al.,* 2012).

De fait, les feature models seront donc amenés à évoluer en fonction de cette communauté de manière agile. L'interface de configuration s'appuyant sur ceux-ci devra donc être suffisamment souple pour pouvoir s'adapter automatiquement à ces changements réguliers.

Nos travaux sont donc guidés par deux préoccupations : (i) la compréhension (*l'utilisabilité*) de l'interface graphique de configuration par un utilisateur non-expert et (ii) une génération de cette interface graphique compatible avec l'évolution fréquente des feature models sous-jacents. Le point (i) est détaillé dans la partie 3 puis nous présentons notre solution évolutive dans la partie 4.

3. Requis pour un utilisateur non-expert

Une première étude de l'outil de configuration supportant la construction de diffuseurs d'informations nous a conduit à identifier selon les lignes de produits différents paradigmes récurrents de configuration.

3.1. *Des features discriminants pour l'utilisateur*

L'expérience nous montre que tous les features ne jouent pas le même rôle en terme de configuration d'un point de vue utilisateur.

2. La mise en place de la communauté est poussée par les étudiants qui au fil du temps ont été associé au projet, et les besoins de nos partenaires industriels qui souhaitent pouvoir enrichir la ligne avec leur propre "produits".

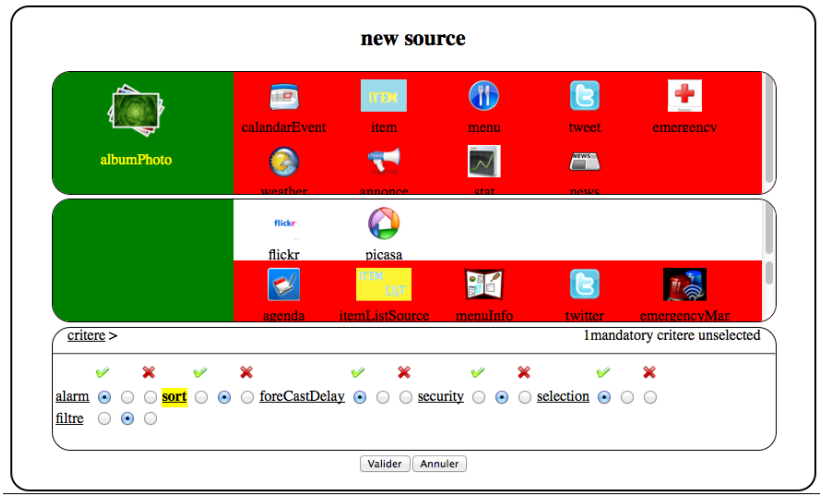

Figure 2. *Capture d'écran de l'IHM de configuration*

Des niveaux de sélection La sélection des sources d'information conduit à privilégier la sélection du type d'information (albums photo, tweets, ...) qui est fortement discriminante, puis à s'intéresser ensuite aux familles de service (par exemple picasa, twitter). Les autres features (critère de sélection des informations, niveau de sécurité, ...) se situent à un niveau plus bas de discrimination et sont généralement sélectionnées suite aux sélections de plus haut niveau.

A contrario, certains feature models, comme celui de *behaviours*, ne possèdent qu'un seul niveau de sélection, toutes les caractéristiques étant toutes aussi discriminantes les unes que les autres.

Dans le contexte de notre étude nous avons rencontré au maximum trois niveaux de sélections tel qu'on peut en avoir un aperçu dans la figure 2. Cette figure présente le configurateur généré dans le cas du feature model de *sources*. Dans les deux zones du haut, les éléments présents dans une zone rouge ne sont plus disponibles à la sélection par le jeu des contraintes, ceux dans une zone verte ont été sélectionnés, et la zone blanche représente les éléments que l'utilisateur peut encore potentiellement choisir. La zone du bas présente les dernières features permettant d'affiner la personnalisation.

Des features non pertinents à la configuration d'un point de vue utilisateur De plus, certaines features ne sont parfois utilisées que par le jeu des contraintes et ne semblent pas pertinentes à montrer à l'utilisateur : par exemple, certaines propriétés techniques ne parleront aucunement à l'utilisateur mais il est intéressant de les modéliser lors de la conception de la ligne.

3.2. *Visualisations imagées*

Plusieurs des features gagnent à être représentées par des icônes qui sont plus pertinentes que des textes : par exemple les logos des services dans le cas des *sources* (Picasa, Twitter, ...), l'iconification d'une disposition par des rectangles juxtaposés dans le cas des *layouts*, une animation pour visualiser des *behaviours* (défilements, ...), etc.

Certains features peuvent être représentées par une même icône complétée par l'identifiant de la caractéristique (par exemple : "ByTag" ou "ByUrl" dans le cas d'un filtrage ou d'un tri). Cette visualisation des features est particulèrement utile lors des premières étapes de construction d'une ligne de produits.

D'autres features ne trouvent pas facilement de représentation graphique. Nous utilisons alors les identifiants. La figure 2 montre ainsi l'utilisation de logos dans les deux premiers niveaux et l'utilisation d'identifiants dans le dernier niveau.

3.3. *Explications*

Lors de la définition des features dans les lignes de produits, il est fréquent de vouloir associer des explications pour à la fois faciliter la maintenance de la ligne elle-même et aider la selection des features par l'utilisateur final. Il s'agira par exemple d'expliquer qu'une source d'information "hyperplanning" diffuse des calendriers ou qu'un type d'information Calendrier respecte le format ical, etc. Ces explications pourront servir aussi bien à décrire plus finement le rôle des features, leur mise en oeuvre, ou encore à expliciter certaines dépendances.

4. Interfaces graphiques de configuration adaptables

Sur la base des exigences liées à notre projet et identifiées dans la partie 3, nous travaillons à la mise en place d'interfaces graphiques de configurations adaptables par des fichiers de configuration.

4.1. *Annotation de FMs et séparation des préoccupations*

Nous constatons tout d'abord que la structure des FMs n'est pas déterminante pour obtenir des informations sur la discrimination des features en niveaux de visualisation. Dans le cas des sources (cf. figure 3 (a))[3], par exemple, le feature model présente essentiellement deux branches, qui ne correspondent pas aux deux principaux niveaux de visualisation. Par ailleurs, il semble difficile et incohérent de forcer l'utilisation d'une structure au sein d'un feature model afin de le faire correspondre à une IHM.

3. Afin de simplifier la lecture des feature models nous avons choisi de ne pas représenter les contraintes tranverses.

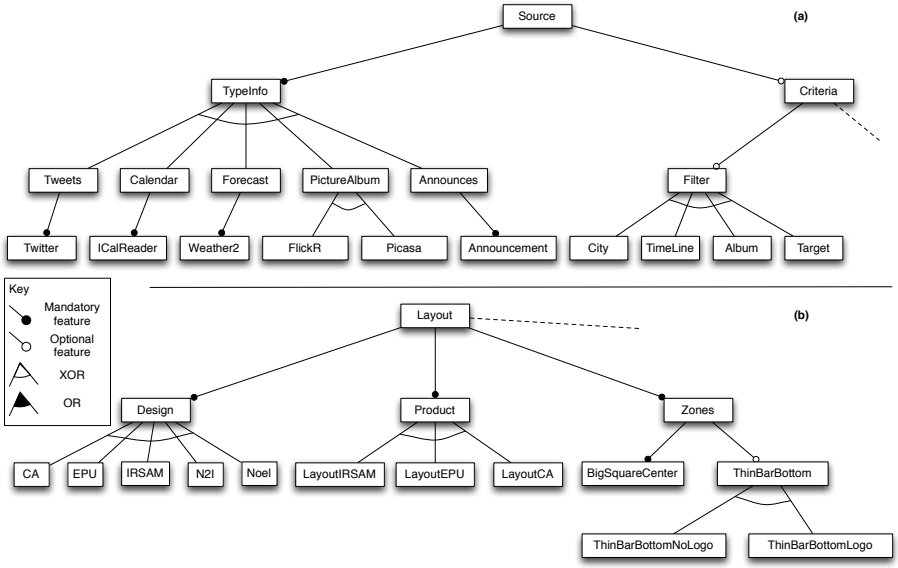

Figure 3. *Représentation simplifiée des feature models de* source *et de* layout

Dans le cas des layouts (cf. figure 3 (b)), le feature model a une structure qui lui est propre, très éloignée de la structure du feature model de sources par exemple.

En outre, rattacher la visualisation à la structure des feature models peut induire des comportements non souhaités lors de leur évolution. Notre expérience sur YourCast nous a ainsi amenés à modifier profondément la structure du feature model de Sources, sans pour autant vouloir modifier les informations et priorités de sélection pour l'utilisateur final. L'évolution du feature model ne doit ainsi pas interférer avec l'interface de configuration présentée à l'utilisateur.

De manière symétrique, il est probable que l'interface de configuration évolue en fonction des retours des utilisateurs, sans pour autant que l'on souhaite modifier les informations contenues dans les features models. Par exemple, dans le cas des behaviours nous considérons pour le moment toutes les informations au même niveau, sans définir de priorité, proposant à l'utilisateur un arbre de critères pour réaliser sa configuration ; il est probable que cette interface évoluera afin de définir des priorités plus aisément compréhensibles par un utilisateur extérieur au domaine, tout en conservant identique notre feature model de behaviours.

Ainsi, il nous semble particulièrement important de conserver une séparation claire entre les features models et la manière dont ils seront représentés dans une interface de configuration. Nous proposons donc un mécanisme d'annotation, utilisable à travers des fichiers de configurations externes aux feature models, permettant d'enrichir ceux-ci automatiquement en leur apportant différentes informations.

4.1.1. *Niveaux*

Le premier besoin auquel doit pouvoir répondre notre système d'annotation consiste à discriminer les features en niveaux. Comme nous l'avons décrit précédemment (cf. 3.1), nous avons identifié dans notre cas d'utilisation au maximum trois niveaux d'affichages et un "niveau" un peu particulier précisant de ne pas afficher le feature.

Nous avons donc défini notre grammaire pour les fichiers d'annotation afin de pouvoir associer un nom de feature à l'un de ces 4 niveaux en utilisant les mots clés suivants :

– *first* : correspond au premier niveau d'affichage, permettant généralement de discriminer grossièrement les produits disponibles ;

– *second* : est associé au second niveau correspondant habituellement aux familles de produits ;

– *criteria* : est associé au troisième niveau dit de critères : il s'agit du seul niveau ayant une forme d'arbre permettant de fixer finement certains choix ;

– *none* : permet de ne pas afficher le feature.

Dans un même fichier d'annotation, il est possible d'utiliser n'importe quelle combinaison de niveaux : leur dénomination n'implique aucune dépendance entre eux. Ainsi, si un utilisateur de l'outil souhaite afficher un niveau de discrimination puis un arbre de sélection, il utilisera les annotations *first* et *criteria*. Il n'y a, à l'heure actuelle, pas d'intérêt à utiliser le niveau *second* lorsque le niveau *first* n'est pas employé. On peut cependant imaginer sans peine des évolutions où son utilisation indépendante ferait sens dans le design de l'interface de configuration par exemple.

Les niveaux peuvent s'appliquer aussi bien sur les features feuilles que sur les noeuds : lorsqu'un niveau est appliqué à un noeud, les noeuds enfants hériteront mécaniquement de la même valeur de niveau. Cependant il est possible de redéfinir la valeur d'un noeud enfant en lui assignant son propre niveau dans le fichier d'annotation. Enfin, lorsqu'une feature ne possède aucune information de niveau de manière explicite ou de manière implicite par l'un de ses noeuds parents, elle prend le niveau *none* par défaut.

4.1.2. *Logos*

Conformément au requis exprimé dans la sous-partie 3.2, notre système d'annotation doit également pouvoir permettre d'associer des images aux features afin de rendre l'interface plus intuitive à l'utilisateur.

La mise en œuvre a consisté ici à enrichir la grammaire de nos annotations afin de permettre d'associer aux features une URL, correspondant à l'image que l'on souhaite afficher pour cette feature. De la même façon que pour les niveaux, un logo associé à un noeud sera automatiquement répercuté à ses enfants si ceux-ci n'en disposent pas eux-même. Enfin, si aucun logo n'est associé à une feature, celle-ci sera affichée au sein de l'interface de configuration avec une image par défaut.

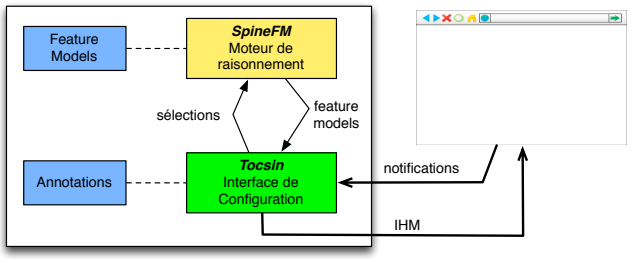

Figure 4. *Architecture globale du système*

Nous avons par ailleurs pu rencontrer lors de notre mise en œuvre d'importantes difficultés à trouver le ou les logos adéquats pour certaines features. Ainsi, dans la majorité des cas, les seuls logos que nous avons attribués concernent les features des niveaux *first* et *second*. En outre, dans certains cas (*renderers*, *behaviours* ...), nous avons envisagé d'utiliser des images animées en guise de logo afin de pouvoir les distinguer aisément les uns des autres. Cette option doit cependant être davantage étudiée afin de déterminer si la présence d'un grand nombre d'images animées ne risquent pas de nuire à l'ergonomie du configurateur comme souligné dans (Nogier, 2008).

4.1.3. *Explications*

Enfin, au sein de nos annotations, il est possible d'enrichir encore les features en leur associant une ou plusieurs explications (cf. 3.3). La mise en œuvre des explications consiste simplement à ajouter une chaîne de caractères aux features.

4.2. *Architecture d'un configurateur*

L'architecture mise en œuvre pour supporter la mise en place des configurateurs est illustrée par la figure 4. La figure nous montre que l'interface de configuration (nommée Tocsin) communique avec un moteur de raisonnement (appelé SpineFM) opérant sur des feature models. Sur la base d'un feature model et d'un fichier d'annotation, Tocsin génère automatiquement l'interface adaptée afin que l'utilisateur puisse réaliser la configuration qui lui convient.

Les opérations de sélection ou de déselection réalisées par l'utilisateur sont envoyées de manière synchrone à SpineFM qui les applique sur sa propre représentation et calcule le FM résultant suite aux possibles contraintes internes. Ce FM est ensuite à nouveau renvoyé à Tocsin pour mettre à jour l'interface de configuration. Ce processus se répète jusqu'à obtention d'un produit dont la validité est déterminée par le moteur de raisonnement.

L'utilisation d'un processus synchrone dans notre approche résulte d'un choix délibéré afin d'empêcher l'utilisateur de violer des contraintes en réalisant plusieurs

opérations sans faire appel au moteur de raisonnement. Nous pallions l'inconvénient potentiel de la lenteur de l'interface par un couplage fort, au sein d'une application déployée dans un même service, des deux processus. Il va sans dire que d'avantages d'études empiriques vont être réalisées afin de valider l'ergonomie de cette approche.

5. Discussions et autres travaux

Ce travail prend ses racines dans les travaux relatifs aux Lignes de produits et aux interfaces Homme-Machine. Les expérimentations menées nous ont confronté à différents problèmes présentés ci-après.

Dé-sélection de features La partie graphique reflète le contenu de la ligne. La dé-sélection d'un feature peut entraîner la dé-sélection d'autres features. C'est la ligne elle-même qui gère les contraintes via SpineFM. L'interface graphique visualise alors uniquement les features sélectionnables ou obligatoires. Ce choix permet de simplifier la configuration lorsque le nombre de features potentiels est grand. Néanmoins la disparition de features peut perturber l'utilisateur, car cela va à l'encontre de l'homogénéité (ou la cohérence) de l'interface graphique (Bastien *et al.,* 1993). Nous envisageons la mise en place de règles qui permettraient d'adapter l'IHM en fonction du nombre de features potentiels.

Explications par déduction Dans (Frey *et al.,* 2012), les auteurs soulignent l'importance pour les utilisateurs d'obtenir des explications sur le comportement de l'IHM elle-même. L'ajout d'explications exploitant les contraintes entre features pour présenter les produits est un point qui devrait être pris en charge par l'IHM (Botterweck *et al.,* 2009, Paul *et al.,* 2005) et ainsi de rester le plus comptatible possible avec le vocabulaire de l'utilisateur (Bastien *et al.,* 1993).

Vocabulaires Lors de la définition des différents FM, nous avons fait des choix de vocabulaire en unifiant ou distinguant des termes, qu'il serait utile de mémoriser pour faciliter la compréhension de la ligne en compléments des explications. L'utilisation d'ontologies pourrait être une façon de traiter ces questions (Czarnecki *et al.,* 2006, Johansen *et al.,* 2010).

Multi-niveaux de configuration Pour l'instant, nous considérons les configurations dans le contexte global de la ligne. A l'instar des auteurs de (Czarnecki *et al.,* 2005), le besoin d'offrir différents niveaux de configuration est apparu pour, par exemple, offrir des outils de configuration n'offrant qu'un choix limités à certaines sources d'informations dans le contexte de l'organisation de rassemblements nécessitant de la diffusion d'informations. Ce point devrait être étudié dans l'année à venir.

6. Conclusion et perspectives

Si les approches génériques en terme d'IHM sont décriées, la définition d'interfaces sur mesure reste une tâche coûteuse et fastidieuse. Des travaux sont menés dans le cadre des lignes de produits afin de permettre la réalisation d'IHM de configuration utilisables pour des modèles de variabilité assez généraux (Boucher *et al.,* 2012). Ces IHM demandent cependant une certaine connaissance dans le domaine des lignes de produits.

Nous proposons une démarche intermédiaire outillée visant à réaliser des IHM de configuration spécifiques à certains feature models, mais automatiquement adaptables aux évolutions qu'ils peuvent ainsi suivre. Dans cette optique, nous avons abstrait certaines propriétés, comme les niveaux de priorité, l'utilisation de logos et les explications. Dans le cadre du projet ANR YourCast, qui vise à faciliter la création de systèmes de diffusion par des techniques de lignes de produits logiciels, ces propriétés nous ont semblé intéressantes à manipuler afin de définir une interface de configuration facile à prendre en main par un utilisateur. Cette abstraction se concrétise par des annotations définies séparément des feature models afin de conserver une bonne séparation des préoccupations. Notre outillage génère ensuite automatiquement l'interface adaptée à partir de ces informations. Lors des sélections de configuration, cette interface communique de manière synchrone avec un moteur de raisonnement sur les feature models.

Les travaux que nous présentons ici consistent en un premier noyau que nous allons développer de manière incrémentale en fonction des besoins rencontrés. Ainsi, nous envisageons dès à présent un certain nombre d'évolutions. La grammaire des annotations, tout d'abord, peut être améliorée afin d'apporter plus de flexibilité dans le traitement des feature models. Donner la possibilité de ne pas associer à la hiérarchie d'un noeud le même niveau et/ou le même logo, par exemple, est une fonctionnalité dont nous ressentons dès à présent le besoin. De même, nous envisageons d'étendre le mécanisme d'explications aux contraintes afin de pouvoir expliquer à l'utilisateur la raison de l'action d'une contrainte, ce qui le conduira à mieux comprendre l'impact de ses choix.

Enfin, la validation de nos travaux a été effectuée par des personnes ayant à la fois une grande connaissance du domaine des Lignes de produits logiciels et notre cas d'utilisation. Afin de valider et d'améliorer notre approche, une étude va être menée auprès d'utilisateurs n'ayant aucune connaissance sur les lignes de produits logiciels, et ce avec l'intervention d'ergonomes.

Remerciements

Les auteurs remercient Laurence Duchien et Daniel Romero pour leur participation au projet yourcast, leurs remarques et nos discussions toujours constructives.

7. Bibliographie

Bastien J. C., Scapin D. L., Ergonomic criteria for the evaluation of human-computer interfaces, Technical Report n° RT-0156, INRIA, June, 1993.

Bosch J., Högström M., « Product Instantiation in Software Product Lines : A Case Study », *in* , G. Butler, , S. Jarzabek (eds), *Generative and Component-Based Software Engineering*, vol. 2177 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 149-163, 2001.

Botterweck G., Janota M., Schneeweiss D., « A design of a configurable feature model configurator », *VAMOS 2009*, 2009.

Boucher Q., Perrouin G., Heymans P., « Deriving configuration interfaces from feature models : a vision paper », *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, ACM, New York, NY, USA, p. 37-44, 2012.

Coutaz J., *Interfaces homme-ordinateur : conception et réalisation*, Dunod, 1990. 455 pages.

Czarnecki K., Helsen S., Eisenecker U., « Staged configuration through specialization and multilevel configuration of feature models », *Software Process : Improvement and Practice*, vol. 10, n° 2, p. 143-169, 2005.

Czarnecki K., Kalleberg K., « Feature Models are Views on Ontologies », *10th International Software Product Line Conference (SPLC'06)*, vol. 1, p. 41-51, 2006.

Frey A. G., Calvary G., Dupuy-Chessa S., « Users need your models ! Exploiting Design Models for Explanations », *Proceedings of HCI 2012, Human Computer Interaction, People and Computers XXVI, The 26th BCS HCI Group conference (Birmingham, UK)*, 2012.

Johansen M., Fleurey F., Acher M., « Exploring the Synergies Between Feature Models and Ontologies », *International Workshop on Model driven Approaches in Software Product Line Engineering MAPLE 2010/SPLC10*, vol. 2, 2010.

Myers B., Hudson S. E., Pausch R., « Past, present, and future of user interface software tools », *ACM Trans. Comput.-Hum. Interact.*, vol. 7, n° 1, p. 3-28, March, 2000.

Nogier J., *Ergonomie du logiciel et design web - 4ème édition - Le manuel des interfaces utilisateur : Le manuel des interfaces utilisateur*, Etudes et développement, Dunod, 2008.

Paul K., Böckle G., Linden F. J., *Software product Line Engineering : Foundation, Principles and Techniques*, hardcover edn, Springer, 2005.

Urli S., Blay-fornarino M., Collet P., « Using Composite Feature Models to Support Agile Software Product Line Evolution », *Models and Evolution*, 2012.

ANNEXE POUR LE SERVICE FABRICATION
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :

Journée Lignes de produits 2012

2. AUTEURS :

Simon Urli, Guillaume Perez, Heytem Zitoun, Mireille Blay-Fornarino, Philippe Collet, Philippe Renevier-Gonin

3. TITRE DE L'ARTICLE :

Vers des interfaces graphiques flexibles de configuration

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :

Vers des IHM flexibles de configuration

5. DATE DE CETTE VERSION :

30 octobre 2012

6. COORDONNÉES DES AUTEURS :

– adresse postale :
Laboratoire I3S (UNS - CNRS)
Les Algorithmes - Bât Euclide B
2000 route des Lucioles – B.P. 121
F-06903 Sophia Antipolis Cedex
prenom.nom@unice.fr
– téléphone : 00 00 00 00 00
– télécopie : 00 00 00 00 00
– e-mail : Roger.Rousseau@unice.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :

LaTeX, avec le fichier de style `article-hermes.cls`,
version 1.2 du 2000/12/01.

8. FORMULAIRE DE COPYRIGHT :

Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
`http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com